isaspec

XML Based ISA Specification

Rob Clark
XDC2022

Motivation

- Replace separate hand-code instruction coding
  - Had grown more complex over gens as new things added
  - Detection of unexpected bits was ad-hoc and incomplete
- Rigorous
- Flexible enough to maintain existing asm syntax
  - Validate conversion via diff
- Flexible enough to handle weird cases
  - Some instructions change across gens
  - Bits repurposed, _HI bits, etc
- Avoid duplication for similar instructions
- IR agnostic

Design

- ISA definition consists of:
  - `<bitset>` – the basic building block
  - `<enum>` – maps value to name/string
  - `<expr>` – simple, named, ‘C’ expression
- Instructions defined as bitsets
  - Arbitrary but fixed length
  - Inheritance to reduce copy/paste
  - Leaf nodes define set of patterns to match

<bitset>

- Bitsets contain
  - `<display>` – template for displaying decoded instruction
  - `<field>` – type can be another `<bitset>`
  - `<derived>` – field derived from expression
  - `<pattern>` – `0`, `1`, `x` (dontcare)
  - `<override>` – mnemonics, etc
  - `<gen>` – restrict bitset to range of generations
  - `<encode>` – maps IR to `<field>`s

```
<bitset name="min.f" extends="#instruction-cat2-2src">
    <pattern low="53" high="58">000001</pattern>
</bitset>
```

<display>

- Template string for disassembly
  - `{FOO}` replaced with decoded value of `FOO`
  - `{NAME}` – special case, replaced with matched bitset name

```
<display>
    {SY}{SS}{JP}{UL}movmsk.w{W} {DST}
</display>
```

<field>

- Field attributes
  - `pos` or `high`/`low` – what bit(s) does the field cover
  - `name`
  - `type`
  - `display` – for `bool` types
- Field elements
  - `<param>` – for bitset types

```
<field name="SAT" pos="42" type="bool" display="(sat)"/>

<field name="SRC0" low="0" high="7" type="#cat1-multi-src">
    <param name="HALF"/>
</field>
```

<derived>

- Derived element attributes
  - `width` – for `int`, to handle sign extension
  - `name`
  - `type`
  - `expr` – expression to derive value

```
<derived name="W" type="uint">
    <expr>
        ({REPEAT} + 1) * 32
    </expr>
</derived>
```

<override>

```
<bitset name="#instruction-cat2-1src" extends="#instruction-cat2">
    <override expr="#cat2-cat3-nop-encoding">
        <display>
            {SY}{SS}{JP}{SAT}(nop{NOP}) {UL}{NAME} {EI}{DST_HALF}{DST}, {SRC1}
        </display>
        <derived name="NOP" expr="#cat2-cat3-nop-value" type="uint"/>
        <field name="SRC1" low="0" high="15" type="#multisrc">
            <param name="ZERO" as="SRC_R"/>
        </field>
    </override>
    <display>
        {SY}{SS}{JP}{SAT}{REPEAT}{UL}{NAME} {EI}{DST_HALF}{DST}, {SRC1}
    </display>
    <field name="SRC1" low="0" high="15" type="#multisrc">
        <param name="SRC1_R" as="SRC_R"/>
    </field>
</bitset>
```

<encode>

```
<bitset name="#instruction" size="64">
     <encode type="struct ir3_instruction *" case-prefix="OPC_">
          ...
     </encode>
</bitset>

<bitset name="stg" extends="#instruction-cat6-stg">
     ...
     <derived name="OFF" width="13" type="offset">
          <expr>({OFF_HI} &amp;lt;&amp;lt; 8) | {OFF_LO}</expr>
     </derived>

     <field   low="9"  high="13" name="OFF_HI" type="int"/>
     <field   low="32" high="39" name="OFF_LO" type="uint"/>

     <encode>
          <map name="OFF_LO">extract_reg_iim(src->srcs[1]) &amp;amp; 0xff</map>
          <map name="OFF_HI">extract_reg_iim(src->srcs[1]) >> 8</map>
     </encode>
</bitset>
```

TODO

- Convert ir2 (a2xx ISA)
  - Fixed length 96b encoding
  - Instructions combine a vec4 + scalar opc
- Convert afuc
  - a5xx μ-code – the CP firmware
  - Easier/cleaner handling of a5xx/a6xx/a7xx differences
  - But need an IR out of disasm (for emulator)
    - Run disasm output back thru existing lexer/parser?
- Other drivers?